INTERNATIONAL JOURNAL OF TECHNOLOGICAL EXPLORATION AND LEARNING (IJTEL)
www.ijtel.org

Max Min Sorting Algorithm

A New Sorting Approach

Smita Paira
B.Tech 2nd Yr Student,
Department of Computer Sc & Engineering,
Calcutta Institute of Technology
Kolkata, India

Sk Safikul Alam
Assistant Professor
Department of Computer Sc & Engineering
Calcutta Institute of Technology
Kolkata, India

Sourabh Chandra
Assistant Professor
Department of Computer Sc & Engineering
Calcutta Institute of Technology
Kolkata, India

Subhendu Sekhar Patra
Assistant Professor
Department of Computer Sc & Engineering
Calcutta Institute of Technology
Kolkata, India

***Abstract*— There are many popular problems in different practical fields of computer sciences, database applications, etc. One of these basic operations is the sorting algorithm. It has attracted a great deal of research. A lot of sorting algorithms has already been developed. These algorithms have enhanced the performance in terms of computational complexity, taking into consideration some factors like time complexity, stability, space complexity, etc. Information growth rapidly in our world leads to increase developing sort algorithms. This paper deals with introducing a new sorting technique that will try to overcome some of the common drawbacks of the basic conventional algorithms.**

***Keywords- Max Min Algorithm; Bubble Sort; Selection Sort; Insertion Sort; Time Complexity*.**

I. INTRODUCTION

We all know, sorting means arranging the elements of an array in ascending or descending order, as per necessity. Such sorting processes are based on some algorithms that need to be executed effectively within a finite amount of time. Based on these algorithms, we find out the time and space complexities of various sorting processes.

There are three conventional sorting algorithms—Bubble sort, Selection sort and Insertion sort. They, all, follow some basic rules and procedures. But all programming languages demand a flexible execution of the algorithms. The above sorting processes perform their operations in one way only i.e. from a particular end of an array. So, the main objective is to overcome this problem/drawback and introduce a new technique.

In this connection, the Max Min algorithm comes into play. This algorithm performs the sorting process from both end of the array. In each step of iteration, it finds out the smallest and largest elements of the array and place them in appropriate positions. Leaving those sorted positions, the execution starts from the remaining elements of the array. Hence, this algorithm takes half the time, which the other algorithms take.

Suppose, there are n elements of an array, then the outer loop starts from 0 to ((n/2)-1).The inner loop starts from i (i.e. the first looping variable) to (n-i-1).Then, it checks for the largest element in the list, taking a temporary variable k and places it in its proper position. The next iterative step of the outer loop sorts out the next smallest and largest elements among the remaining list. This process continues until all the elements are sorted. For example, let us take an array of 10 elements. There will be 5 passes. These passes are as follows:-

Let the original array be

3	7	8	5	11	2	1	9	6	4
0	1	2	3	4	5	6	7	8	9

Pass 1: The algorithm scans for the largest element (11) and smallest element (1) and put them in the proper place as shown below.

1	4	7	5	8	3	2	9	6	11
0	1	2	3	4	5	6	7	8	9

Pass 2: Leaving 1 and 11, the algorithm scans the remaining list, finds out the largest element (9) and smallest element (2) and puts them in the proper place as shown below.

1	2	6	5	7	4	3	8	9	11
0	1	2	3	4	5	6	7	8	9

Pass 3: Leaving 1, 2, 9 and 11, the algorithm scans the remaining list, finds out the largest element (8) and smallest element (3) and puts them in the proper place as shown below.

1	2	3	6	7	5	4	8	9	11
0	1	2	3	4	5	6	7	8	9

Pass 4: Leaving 1, 2, 3, 8, 9 and 11, the algorithm scans the remaining list, finds out the largest element (7) and smallest element (4) and puts them in the proper place as shown below.

1	2	3	6	7	5	4	8	9	11
0	1	2	3	4	5	6	7	8	9

Pass 5: Now the algorithm scans for positions 4 and 5, and sorts them as follows.

1	2	3	4	5	6	7	8	9	11
0	1	2	3	4	5	6	7	8	9

Hence, the sorted array is as follows:-

1	2	3	4	5	6	7	8	9	11
0	1	2	3	4	5	6	7	8	9

II. PSUEDO CODE

The pseudo code of the Max Min Algorithm is as follows:-

Step 1: Initialize k to (n-1)

Step 2: Initialize i to 0

Step 3: Initialize j to i

Step 4: If the value at position j is greater than the value at position k, then swap the two elements.

Step 5: If the value at position i is greater than the value at position j then swap the two elements.

Step 6: Follow step 4 and step 5 until j becomes (n-i-1).Then print the list for that particular pass (i+1)

Step 7: Decrement k by 1

Step 8: Follow step 3-step 7 until i becomes ((n/2)-1)

Step 9: Finally, print the sorted array

Step 10: Exit.

III. IMPLEMENTATION OF MAX MIN ALGORITHM USING JAVA AND C

Let's see how the pseudo code can be realized using object oriented programming approach and a function oriented programming language as well.

A. The Max Min Sorting in Java is as follows

```
import java.io.*;
class sort
{
public static void main(String args[])throws IOException
{
BufferedReader buf=new BufferedReader(new
InputStreamReader(System.in));
  int a[]=new int[10];
  System.out.println("Enter the number of elements :-");
  int n=Integer.parseInt(buf.readLine());
  int i,j;
  System.out.println("\n Enter the array elements :-");
        for(i=0;i<n;i++)
          a[i]=Integer.parseInt(buf.readLine());
        System.out.println("\n The unsorted array is :- \n");
        for(i=0;i<n;i++)  //printing the unsorted array
            System.out.print(a[i]+" ");
            max_min(a);  //calling the Max Min function
            System.out.println("\n The sorted array is :-
            \n");
            for(i=0;i<n;i++)  //printing the sorted array
                System.out.print(a[i]+" ");
                static void max_min(int b[])
                {
                  int tmp,l,k=n-1;
                  for(i=0;i<(n/2);i++)
                  {
                  for(j=i;j<(n-i);j++)
                  {
                  if(b[j]>b[k]) //finding the maximum
                                element
                        {
                        tmp=b[j];
                        b[j]=b[k];

        b[k]=tmp;
        }
        if(b[i]>b[j]) //finding the minimum element
                {
                tmp=b[i];
                b[i]=b[j];
                b[j]=tmp;
                }
        } //end of  j-loop

System.out.println(" \n The array after pass "+(i+1)+" is :-
```

```
                \n");
for(l=0;l<n;l++) //printing the array after every pass
System.out.print(b[l]+" ");
k--;
} //end of  i-loop
} //end of function max_min()
} //end of public....
} //end of class sort
```

B. The Max Min Sorting in C is as follows

```
#include<stdio.h>
#include<conio.h>
void max_min(int *,int);
void main()
{
        int a[10],i,n;
        clrscr();
        printf("\n Enter the number of elements....");
        scanf("%d",&n);
        printf("\n Enter the elements....");
        for(i=0;i<n;i++)  //taking input from user
        {
                scanf("%d",&a[i]);
        }
        printf("\n The  unsorted array is....\n");
        for(i=0;i<n;i++)   //printing the unsorted array
        {
                printf("%d ",a[i]);
        }
        max_min(a,n);
        printf("\n The sorted array is....\n");
        for(i=0;i<n;i++) //printing the sorted array
        {
                printf("%d ",a[i]);
        }
        getch();
} //end of main()
void max_min(int *b,int m)
{
        int k,l,i,j,tmp;
        k=m-1;
        for(i=0;i<(m/2);i++)
        {
                for(j=i;j<(m-i);j++)
                {
                        if(b[j]>b[k]) //finding the
maximum element
                        {
                                tmp=b[j];
                                b[j]=b[k];
                                b[k]=tmp;
                        }
                        if(b[i]>b[j])  //finding the minimum
element
                        {
                                tmp=b[i];
                                b[i]=b[j];
                                b[j]=tmp;
                        }
                } //end of j-loop
                printf("\n The array after pass %d
is....\n",i+1);
                for(l=0;l<m;l++)  //printing the array after
every pass
                {
                        printf("%d ",b[l]);
                }
                k--;
        } //end of i-loop
} //end of max_min()
```

IV. TIME COMPLEXITY

Let the number of elements in the array be n. For first pass, the inner for-loop iterates n times. For second pass, the inner for-loop iterates (n-2) times, and so on. Now, if the number of elements is even, then the time complexity is

$$=n+(n-2)+(n-4)+\ldots\ldots\ldots\ldots\ldots\ldots+2$$
$$=(n/4)[2n + ((n/2)-1)(-2)]$$
$$=(n/4)(2n-n+2)$$
$$=(n(n+2))/4$$
$$=(n^2/4)+((2n)/4)$$
$$=O(n^2)$$

Now, if the number of elements is odd, then the time complexity is

$$=n+(n-2)+(n-4)+\ldots\ldots\ldots\ldots\ldots\ldots+3$$
$$=((n-1)/4)[2n + (((n-1)/2)-1)(-2)]$$
$$=((n-1)/4)(2n-n+1+2)$$
$$=((n-1)(n+3))/4$$
$$=(n^2+2n-3)/4$$
$$=O(n^2)$$

Hence, the overall time complexity is $O(n^2)$. Now if we consider for the best case and worst case, we will find that in both the cases, the outer loop will execute n/2 times and the inner loop will execute (n-i) times. Hence, the time complexity is again $O(n^2)$ for best and worst cases both.

V. COMPARISON ANALYSIS

Now, we will discuss here a short comparison between the newly suggested algorithm and the other conventional algorithms like Bubble sort, Selection sort, and Insertion sort. Bubble sort has the capability to do something that most sorting algorithms cannot. It passes through the list until no swaps are done thus indicating the array being sorted [2,3]. Hence, its best case time complexity is O(n).In terms of swaps,

Bubble sort is worse than Selection sort as it has $O(n^2)$ swaps and the latter has O(n) swaps[3]. But, the number of comparisons are same [9]. Even if the array is sorted, the Selection sort takes more time than Bubble sort [10]. Both Bubble and Insertion sorts have same number of swaps and comparisons [3,8]. If we analyze the Max Min Algorithm, we will find that the number of swaps in the best case and worst case are 0 and (n/2) respectively. As a result, its efficiency is high.

Insertion sort is faster than the other two sorting algorithms [6,8] but is expensive as it requires shifting of all elements over by one [10,2]. All four algorithms, including the Max Min Algorithm, are simple [1,5] and requires less space in memory as no additional storage is required [5,2]. In Bubble Sort, Selection Sort and Insertion Sort, we can sort the items from one end only whereas in the Max Min Algorithm, we sort from both the ends and move to the middle.

Both Bubble and Selection sort, on each pass, it finds the maximum and minimum elements respectively and leaving that particular position, performs on the rest of the array [4,8]. But the Max Min algorithm finds both the maximum and minimum elements on a single pass and leaving those two positions, works on the rest of the list in the next pass. Thus, it is more efficient and advantageous.

One major drawback behind the three sorting algorithms, namely Bubble sort, Selection sort and Insertion sort is that they are very slow when the list is very large[1,7]. Hence, they do not find application in real life [5, 10] and are mainly used for educational purposes [1,8]. In this case also, the Max Min Algorithm is advantageous. Even if the list is large, it performs the sorting operation very efficiently and fast, with minimum number of comparisons required. It takes ((n(n+2))/4) comparisons for even numbers of elements and $((n^2+2n-3)/4)$ comparisons for odd number of elements which is less than the ((n(n-1))/2) comparisons, as required by Bubble sort, Selection sort and Insertion sort. Thus, the Max Min algorithm is more effective rather than other conventional sorting algorithm and has many real-life applications. We compared the MAX MIN Sorting, with other sorting algorithms like BUBBLE Sort, INSERTION Sort, and the SELECTION sort in a laptop with Core2 Due 2.1Ghz Processor, 4 GB 800 Mhz FSB DDR2 RAM on Windows-7 Platform and found the comparison study as stated below.

TABLE I. COMPARISON STUDY OF DIFFERENT SORTING ALGORITHM

Conventional Sorting Algorithm	Time Complexity	Total No of Comparison required (Unsorted Array Size of 20,50 ,100 and 200 respectively)	Execution Time in Worst case (Unsorted Array Size of 20,50 ,100 and 200 respectively)
MAX MIN Sort	O(n2)	a) 110 b) 650 c) 2550 d) 10100	a) 1.098901099ms b) 1.428571429 ms c) 1.868131868 ms d) 2.252747253 ms
BUBBLE Sort	O(n2)	a) 190 b) 1225 c) 4950 d) 19900	a) 1.593406593 ms b) 1.538461538 ms c) 1.923076923 ms d) 3.681318681 ms
INSERTION Sort	O(n2)	a) 210 b) 1275 c) 5050 d) 20100	a) 1.373626374 ms b) 1.428571429 ms c) 2.747252747 ms d) 3.406593407 ms
SELECTION Sort	O(n2)	a) 190 b) 1225 c) 4950 d) 19900	a) 1.208791209 ms b) 1.373626374 ms c) 2.912087912 ms d) 3.461538462 ms

Therefore other than conventional sorting the MAX MIN Sorting is suitable for the unsorted array with large size.

VI. CONCLUSION

This paper deals with the introduction of a new sorting algorithm (i.e. the algorithm) and its analysis .We have also mentioned a comparison study of the three conventional sorting algorithms with the newly proposed algorithm. This research is an initial step for future work. In the future, we will try to improve our algorithm to find out the ways in which it can be applied in various practical and real-life applications. We will also try to extend our ideas to invent more such algorithms which will be helpful for sorting operation as well as software technology.

REFERENCES

[1] Surender Lakra, Divya," Improving the performance of selection sort using a modified double-ended selection sorting", International Journal of Application or Innovation in Engineering & Management (IJAIEM),ISSN: 2319 – 4847, Volume 2, Issue 5, May 2013, pp 364-370.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "*Introduction to Algorithms*", Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Problem 2-2, pg.38.

[3] Augenstein & Tenenbaum Langsam , "*Data Structure Using C & C++"*, 2nd Ed, ISBN13: 9788120311770 Seymour Lipschutz , "Data structure & Algotihm", Schaum's Outlines Tata McGraw Hill 2nd Edition , ISBN13: 9780070991309.

[4] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, " Data Structures & Algothims", 2nd Edition, ISBN13: 9780201000238, Chapter 8, Page: 366-425

[5] Jon Kleinberg, Eva Tardos, " Algorthim Design", First Edition, ISBN13: 9780321372918 ,Chapter 2, Page: 48-58.

[6] Donald Knuth. "The Art of Computer Programming", Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0. Pages 106–110 of section 5.2.2: Sorting by Exchanging.

[7] M. Thorup. "*Randomized Sorting in O(n log log n) Time and Linear Space Using Addition, Shift, and Bit-wise Boolean Operations. Journal of Algorithms"*, Volume 42, Number 2, February 2002, pp. 205-230(26).

[8] Wirth, Niklaus, *"Algorithms and data structures"*, Prentice Hall Publication, ISBN13: 9780130219992,

[9] G.S. Baluja, " *Data Structure through C",* Fourth Edition, ISBN13: 9788174462855, Chapter 10, Page 541-550.

[10] Adamson, Iain T. ,*"Data structures and algorithms: A First Course"*, ISBN13: 9783540760474, Page: 79-85